SafeLoc: Token-Level Hallucination Localization in LLM-Generated Code via CodeBERT

Buwei Wang[1], Liping Wen[1], Hao Jiang[1], Fanglin Xu[1], Zijiao Zhang[1,*]
[1]*Zhengzhou University*, Zhengzhou, China
Email: buwei@gs.zzu.edu.cn, {867222670, 2659762339, 3211086218}@qq.com, zzj@zzu.edu.cn

***Abstract*—Large Language Models (LLMs) have revolutionized the code generation paradigm, yet their inherent "hallucination" issue—generating plausible but logically incorrect code or invoking non-existent libraries—poses severe security risks. Existing detection methods are mainly limited to coarse-grained classification or rely on expensive external execution environments and LLM self-reflection, lacking low-cost and high-precision token-level localization capabilities. To address this, we propose SafeLoc, a fine-grained hallucination localization framework. Utilizing CodeBERT as the semantic backbone, the framework integrates Bidirectional Long Short-Term Memory (BiLSTM) to enhance long-range context modeling and employs a Focal Loss-based training strategy to address the class imbalance caused by sparse hallucination tokens. Experimental results demonstrate that SafeLoc achieves an F1-score of 97.14% on an independent test set. Notably, in terms of micro-localization precision, it achieves a semantic tolerance accuracy (Diff $\leq$ 3) of 86.04%, demonstrating stronger applicability in micro-localization than GPT-5 Mini and Claude 4.5 Haiku. Furthermore, entropy-based uncertainty analysis confirms the interpretability and trustworthiness of the proposed method.**

***Index Terms*—Code Hallucination, Large Language Models, Fine-grained Localization, Sequence Labeling, Uncertainty Estimation**

I. Introduction

With the release of GPT-5.2 and Claude Opus 4.5, Large Language Models (LLMs) have demonstrated exceptional code generation capabilities. However, reliability remains a primary obstacle. **Rawte et al. [1]** identify "factual hallucination" as a persistent issue. **Liu et al. [2]** note that code hallucinations often manifest as fictitious APIs or subtle logic bugs. Empirical studies by **Wang et al. [3]** further reveal that many codes contain security vulnerabilities. As emphasized by **Kumar et al. [4]**, utilizing pre-trained models to proactively identify these risks is critical for securing the software supply chain.

Existing detection methods are limited by strict environmental requirements or high inference overheads. Moreover, metrics like **CodeScore [5]** focus on sequence-level scoring, lacking fine-grained localization. We observe that generative LLMs suffer from an inherent "granularity mismatch" in micro-localization tasks, making it difficult to pinpoint errors and increasing debugging costs.

To address this, we propose **SafeLoc**, a lightweight fine-grained localization framework. Inspired by **Pointer Networks [6]**, SafeLoc models localization as a sequence labeling task, analyzing internal states to achieve token-level localization without external execution.

Our contributions are: (1) We propose the SafeLoc framework, integrating temporal modeling and hard sample mining for real-time localization; (2) SafeLoc achieves an F1-score of **97.14%**, outperforming the general-purpose **RoBERTa [7]** by **8.36%** on hard samples; (3) Comparison with SOTA generative models (GPT-5 Mini, Claude 4.5 Haiku) demonstrates SafeLoc's task-specific superiority in localization precision (Diff $\leq$ 3).

II. Related Work

A. Hallucinations in Code Generation

With the rapid development of deep learning technologies, the performance of code generation models has made significant strides. However, generated code often faces dual challenges regarding accuracy and faithfulness. **Liu et al. [2]** conducted a systematic analysis of code hallucinations, pointing out that models tend to generate code that is syntactically correct but functionally inconsistent with the user's intent. Unlike general text generation, the survey by **Luo et al. [8]** indicates that hallucinations in the code domain are more difficult to measure via simple semantic similarity, as subtle symbolic differences (e.g., variable misuse or operator errors) can lead to severe execution failures. Consequently, research focusing on the specific characteristics of code hallucinations has emerged as a crucial branch of study.

B. Hallucination Detection Approaches

Existing detection methods primarily focus on different validation dimensions:

Execution-based Methods: CodeHalu [9] proposes leveraging the execution results of test cases to verify code correctness. While achieving high accuracy, the applicability of such methods is often severely constrained by the availability of comprehensive test cases and the stringent requirements for configuring execution environments.

Consistency-based Methods: SelfCheckGPT [10] assesses factuality by sampling multiple model responses and calculating inter-sample consistency. Although this approach requires no external knowledge base, the substantial computational overhead introduced by multiple inferences is a non-negligible constraint in real-time application scenarios. Furthermore, research by **Chen et al. [11] (FELM)** has also explored

*Corresponding author.

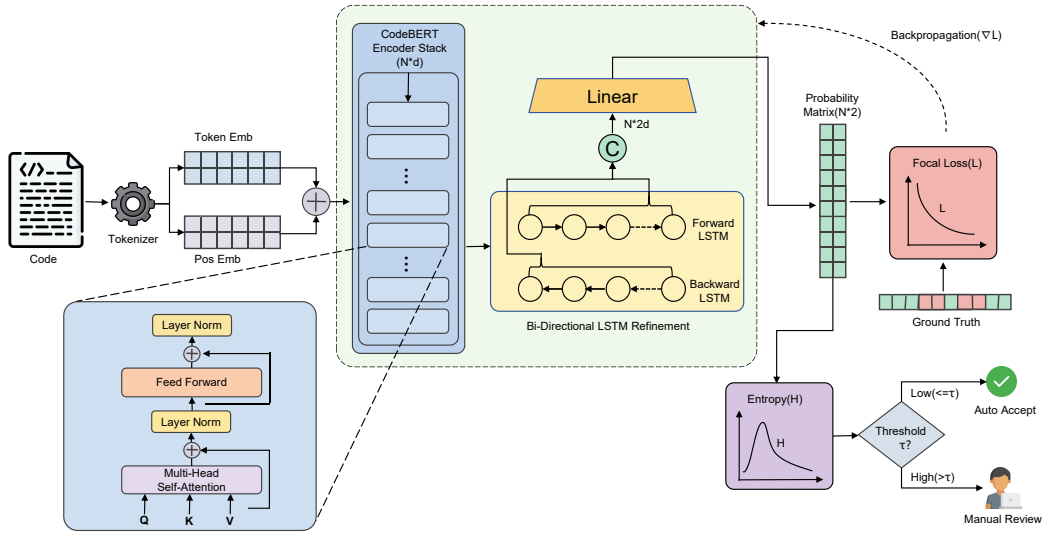

Fig. 1. The overall architecture of SafeLoc. It leverages CodeBERT for semantic encoding and integrates a BiLSTM-based context refinement layer to capture long-range dependencies. Finally, the classification head outputs token-level hallucination probabilities optimized by Focal Loss.

the limitations of utilizing Large Language Models (LLMs) themselves as evaluators.

Reference-based Methods: CodeMirage [12] detects hallucinations by comparing generated code against ground truth solutions. While highly effective in benchmark settings, the application scope of such methods is limited in open-ended generation scenarios where high-quality reference code is often unavailable.

III. Methodology

A. Problem Formulation & Overview

We formulate the code hallucination localization task as a sequence labeling problem. Given a code sequence $X = \{x_1, x_2, \ldots, x_N\}$ generated by an LLM, where N is the sequence length, our goal is to learn a mapping function $f_\theta : X \to Y$ to predict the label sequence $Y = \{y_1, y_2, \ldots, y_N\}$, where $y_i \in \{0, 1\}$ indicates whether the i-th token is a hallucination (1 for hallucination, 0 for normal). As shown in Fig. 1, SafeLoc consists of three core modules: (1) a CodeBERT-based semantic encoder; (2) a BiLSTM-based context refiner; and (3) an optimization and uncertainty estimation module based on Focal Loss.

B. Semantic Encoding Layer

To fully comprehend the syntax and semantics of code, we adopt **CodeBERT [13]** as the backbone network. Unlike RoBERTa, which is trained solely on natural language, CodeBERT is pre-trained on large-scale bimodal data (code-comments), enabling it to better capture structural features of programming languages. Given an input sequence X, we first tokenize it into Token IDs using a Byte-Pair Encoding (BPE) tokenizer, and then feed it into CodeBERT to obtain contextual hidden states H:

$$H = \text{CodeBERT}(x_1, \ldots, x_N) \in \mathbb{R}^{N \times d} \tag{1}$$

where d is the hidden layer dimension ($d = 768$ in this study).

C. Contextual Refinement & Classification

Although Transformers possess global attention mechanisms, capturing local sequential continuity is crucial for identifying error boundaries in sequence labeling. Code often contains long-range "definition-use" chains, and pure self-attention may struggle to sensitively detect such temporal logic breaks. Inspired by **Zhou et al. [14]**, we introduce a **Bidirectional Long Short-Term Memory (BiLSTM)** network on top of the CodeBERT output layer. BiLSTM models the control flow of code from both forward and backward directions, further refining the temporal dependencies between tokens:

$$h'_i = \text{BiLSTM}(h_i), \quad h'_i \in \mathbb{R}^{2d'} \tag{2}$$

where h_i is the hidden state from CodeBERT, h'_i is the feature vector enhanced by bi-directional temporal modeling, and $2d'$ is the output dimension of the BiLSTM. Finally, a linear classification head maps the enhanced features to the binary classification space, and the probability of each token being a "hallucination" is calculated via the Softmax function:

$$p(y_i = 1|x) = \text{Softmax}(W h'_i + b) \tag{3}$$

D. Optimization Objective

In code hallucination detection, we face a severe class imbalance challenge: hallucination tokens can be sparse in some snippets while dense in others (e.g., truncated errors). Standard Cross-Entropy loss is easily dominated by the gradients of

numerous simple negative samples (normal code), making it difficult for the model to converge to an optimal solution.

To address this, we employ **Focal Loss [15]** as the training objective. By introducing a modulating factor $(1-p_t)^\gamma$, this loss function down-weights easy examples, forcing the model to focus on hard boundary tokens (Hard Negatives):

$$\mathcal{L}_{FL} = -\alpha_t(1-p_t)^\gamma \log(p_t) \tag{4}$$

where p_t is the model's estimated probability for the true class. **Parameter Settings:** In this study, we set the focusing parameter $\gamma = 0.5$ and the balancing factor $\alpha = 1$. We chose $\gamma = 0.5$ based on preliminary experiments indicating that while hallucination boundaries are fuzzy, they are not extremely indistinguishable. A lower γ value allows the model to focus on hard samples while retaining perception of global syntactic structures, preventing overfitting to noise in the early stages of training.

E. Uncertainty Estimation

To enhance the interpretability of the model in safety-critical scenarios, we introduce an uncertainty quantification mechanism. Inspired by **Gal and Ghahramani [16]** on uncertainty representation in deep learning, and combined with the work of **Hendrycks et al. [17]**, we utilize the entropy of the predictive distribution to measure the model's confidence in its current decision:

$$\text{Uncertainty}(x_i) = -\sum_{c\in\{0,1\}} p(y_i=c)\log p(y_i=c) \tag{5}$$

A higher entropy value indicates that the model is hesitant about the current token. In subsequent experiments, we demonstrate that high uncertainty is often highly correlated with prediction errors, providing a quantitative basis for "Human-in-the-Loop" verification.

IV. Experimental Setup

A. Dataset and Metrics

The dataset used in this study is derived from **Collu-Bench [18]**, a high-quality Python code generation dataset with token-level hallucination annotations. To prevent data leakage, we perform data splitting based on **independent task instances**: *Data 1–4* serve as the training set (10,548 samples), while *Data 5* is reserved as a held-out test set (2,637 samples). This ensures that the code logic in the test set is unseen during the training phase.

For evaluation, we employ standard token-level **Precision**, **Recall**, and **F1-score** to measure the overall identification capability. Furthermore, following the recommendations of **Kasner et al. [19]**, we introduce **Start Position Accuracy** to evaluate localization precision:

- **Exact Match (Diff=0):** The predicted starting token index matches the ground truth exactly.
- **Soft Match (Diff $\leq$ 3):** The predicted position falls within a tolerance window of 3 tokens from the ground truth, assessing the localization capability within a semantic tolerance range.

B. Baselines

To comprehensively evaluate the effectiveness of SafeLoc, we select four categories of representative models for comparison:

1) **Untrained CodeBERT:** A model initialized with pre-trained weights but without fine-tuning on the target dataset. It serves as a lower bound to evaluate zero-shot transferability.
2) **RoBERTa-base [7]:** A general-purpose pre-trained language model, used to verify the necessity of code-specific pre-training.
3) **CodeBERT-CE:** A CodeBERT model fine-tuned using standard Cross-Entropy loss and a linear head. It acts as the primary control group to validate the effectiveness of the BiLSTM and Focal Loss components in SafeLoc.
4) **SOTA LLMs:** We employ GPT-5 Mini and Claude 4.5 Haiku to explore paradigmatic differences. We utilized zero-shot prompting to request the exact starting token, aligning responses to the original sequence via string matching and resolving multiple occurrences by proximity to the ground truth.

C. Implementation Details

All experiments were implemented using PyTorch and HuggingFace Transformers. For SafeLoc, we set the BiLSTM hidden size to 256 with a dropout rate of 0.1. The Focal Loss focusing parameter is set to $\gamma = 0.5$. We utilized the **AdamW optimizer [20]** with an initial learning rate of 2e-5 and a batch size of 4, training for 3 epochs. The experiments were conducted on a single **NVIDIA GeForce RTX 3090 GPU**.

V. Results and Analysis

To comprehensively evaluate the performance of SafeLoc and verify the effectiveness of the proposed method, this section discusses the following four research questions (RQs).

A. RQ1: Overall Performance and Localization Precision

We compared SafeLoc with three categories of baselines across all dimensions, as shown in Table I.

TABLE I
MAIN PERFORMANCE COMPARISON ON THE TEST SET. THE BEST RESULTS ARE HIGHLIGHTED IN BOLD.

Model	Precision	Recall	F1-Score	Exact Match (Diff=0)	Soft Match (Diff$\leq$3)
Untrained	68.56%	71.37%	69.94%	0.42%	48.96%
RoBERTa-base	88.44%	97.30%	92.66%	20.14%	61.13%
CodeBERT-CE	89.08%	**98.52%**	93.56%	28.63%	67.50%
SafeLoc (Ours)	**97.29%**	95.77%	**97.14%**	**54.30%**	**86.04%**

Although the generic model **RoBERTa-base [7]** shows a minimal gap in F1-score compared to **CodeBERT-CE** (Baseline), its localization accuracy metrics are consistently lower. This indicates that generic models tend to "fuzzily cover"

large regions to boost F1 scores, lacking precise perception of syntactic boundaries in code.

Compared to CodeBERT-CE, our **SafeLoc** achieves an F1-score of **97.14%** and, more critically, nearly doubles the **Exact Match** to **54.30%**. This proves that our architectural improvements successfully overcome the "localization ambiguity" bottleneck of traditional models, achieving high-precision localization.

Notably, SafeLoc achieves an accuracy of **86.04%** within a semantic tolerance of $k = 3$ (Soft Match). The gap between Exact Match (54%) and Soft Match (86%) is primarily attributed to the inherent characteristics of subword tokenization. As pointed out by **Chai et al. [21]**, the tokenization strategy may cause slight shifts between human-understood semantic boundaries and model token boundaries. Therefore, the high Soft Match of 86.04% strongly evidences that our model genuinely comprehends the causality of code logic and can accurately pinpoint the starting region of hallucinations.

B. RQ2: Ablation Study

We sequentially introduced the Focal Loss and BiLSTM modules starting from CodeBERT-CE. The results are summarized in Table II.

The introduction of **Focal Loss [15]** led to a qualitative leap in performance. Exact Match surged from 28.63% to 50.28%, and Soft Match improved from 67.50% to 84.43%. This proves that standard Cross-Entropy is easily dominated by a large number of simple negative samples in code hallucination tasks, leading to blurred boundary localization. Focal Loss dynamically adjusts weights to force the model to focus on those hard-to-distinguish boundary tokens, thereby significantly enhancing localization "sharpness."

The **BiLSTM** module further pushed Exact Match to **54.30%** and Soft Match to **86.04%**. This indicates that while Focal Loss addresses the "focus" issue, the bi-directional temporal modeling capability of BiLSTM remains indispensable for understanding the contextual flow of code and correcting subtle boundary deviations.

TABLE II
ABLATION STUDY OF COMPONENT EFFECTIVENESS. WE SEQUENTIALLY INTRODUCE FOCAL LOSS AND BILSTM TO THE BASELINE.

Model Variant	Component Added	F1-Score	Exact Match (Diff=0)	Soft Match (Diff≤3)
Baseline	Cross-Entropy Loss	93.56%	28.63%	67.50%
+ Focal	Focal Loss	96.52%	50.28%	84.43%
+ BiLSTM (SafeLoc)	**BiLSTM Layer**	**97.14%**	**54.30%**	**86.04%**

C. RQ3: Comparison with SOTA Generative LLMs

To verify the irreplaceability of specialized discriminative models, we compared SafeLoc with the current state-of-the-art generative models, Claude 4.5 Haiku and GPT-5 Mini (see Fig. 2).

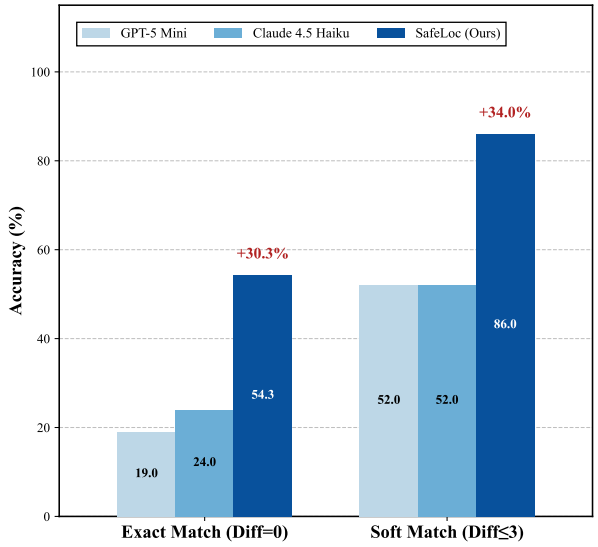

Fig. 2. Localization accuracy comparison between SafeLoc and SOTA LLMs (GPT-5 Mini, Claude 4.5). SafeLoc demonstrates superior precision in both Exact Match and Soft Match, significantly overcoming the granularity mismatch issue inherent in generative models.

As shown in Fig. 2, generative models exhibit significant granularity mismatch problems in micro-localization tasks. Even the best-performing Claude 4.5 achieves an Exact Match of only 24.00%, which is lower than the CodeBERT-CE baseline. GPT-5 Mini performs even worse at 19.00%. In contrast, SafeLoc serves as a highly effective alternative, achieving **54.30%** Exact Match and **86.04%** Soft Match.

This huge performance disparity stems from the fundamental difference in model architectures. LLMs are generated based on an **auto-regressive** mechanism; they tend to generate explanatory natural language or rewritten code, making it difficult to establish a precise mapping from high-level semantic errors to low-level token indices internally. In contrast, SafeLoc adopts a **Sequence Labeling** paradigm, scanning tokens one by one, thus possessing a natural architectural advantage in safety auditing scenarios requiring pixel-level precision.

D. RQ4: Reliability and Qualitative Analysis

To investigate the trustworthiness and interpretability of the model's decisions, we calculated the predictive entropy distribution of the model on the test set (as shown in Fig. 3).

The results reveal significant distributional shifts: when the model localizes accurately, the entropy is extremely low and highly concentrated, indicating strong certainty in correct decisions; when prediction deviations occur, the entropy increases significantly and the distribution widens. This phenomenon is consistent with the findings of **Wang et al. [22]**, confirming that the model's **self-confidence** is an effective indicator for identifying potential errors.

This significant distribution difference indicates that SafeLoc possesses good error awareness. In practical deployment,

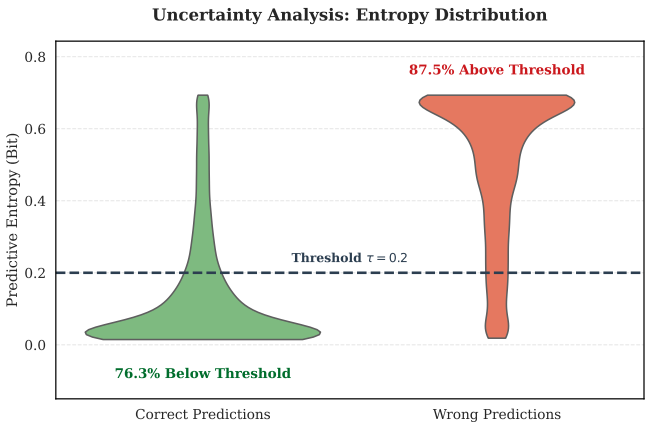

Fig. 3. Uncertainty Analysis: Entropy distribution for correct vs. wrong predictions. The significant distributional shift indicates that the model exhibits higher uncertainty (higher entropy) when making erroneous predictions, validating its reliability for human-in-the-loop verification.

we can set an empirical threshold (e.g., $\tau = 0.2$) to automatically trigger human review when the predicted entropy exceeds this value, thereby constructing a high-security **Human-in-the-Loop** detection system.

VI. Discussion

A. Practical Implications

The core value of this study lies not only in hallucination detection but also in empowering downstream tasks.

First, due to the lightweight architecture of SafeLoc, its inference latency is minimized, making it highly suitable for integration into IDEs as a **Just-in-Time Code Reviewer**. When developers accept code blocks generated by LLMs, SafeLoc can scan and highlight potential hallucinations **with extremely low latency**, establishing a first line of defense for code security.

Second, existing Automated Program Repair (APR) tools are often limited by inaccurate fault localization, resulting in excessively large search spaces. The precise **token-level boundaries** provided by SafeLoc can serve as "anchors" for repair models, locking the repair scope within the minimal semantic unit. As stated by **Hossain et al. [23]**, precise fault localization is a key prerequisite for significantly improving repair success rates and efficiency.

B. Threats to Validity & Limitations

To ensure academic rigor, we analyze the potential limitations of this study:

- **Data Distribution Bias:** Our experiments are primarily based on Python datasets. Although code structures share generalities, performance on strongly typed languages like Java or C++ requires further verification.
- **Long Context Limitation:** Constrained by the pre-training architecture of CodeBERT, SafeLoc has a maximum input length of 512 tokens. This implies that the current version is mainly applicable to function-level or snippet-level hallucination detection. For ultra-long code, we employed truncation, which may result in the loss of some contextual information. Future work will consider adopting Longformer or sliding window mechanisms to overcome this restriction.
- **Metric Applicability:** Comparisons with generative LLMs should be interpreted with caution. Their auto-regressive nature inherently disadvantages them in discriminative token-level metrics compared to our task-specific architecture.

VII. Conclusion and Future Work

This paper proposes **SafeLoc**, a fine-grained hallucination localization framework for LLM-generated code. Addressing the challenges of high concealment and detection costs associated with code hallucinations, we designed a discriminative architecture that fuses **BiLSTM temporal modeling** with **Focal Loss-based hard sample mining**.

Experimental results demonstrate that SafeLoc achieves superior performance while maintaining low computational costs. Particularly in micro-localization tasks, its positional accuracy significantly outperforms SOTA generative large models (GPT-5 Mini and Claude 4.5 Haiku). Furthermore, uncertainty analysis confirms the interpretability and trustworthiness of the model when facing prediction deviations.

In future work, we plan to: (1) Reference the visual evaluation framework of **LLMMaps [24]** to extend SafeLoc to broader evaluations across multiple languages (e.g., Java, C++) and tasks; (2) Integrate SafeLoc's localization capability with **Retrieval-Augmented Generation (RAG)**, inspired by **Tao et al. [25]**. By using the hallucinated tokens localized by SafeLoc as retrieval anchors to fetch correct API documentation from external knowledge bases, we aim to build a closed-loop "Detect-Repair" system.

References

[1] V. Rawte, S. Chakraborty, A. Pathak, A. Sarkar, S. T. I. Tonmoy, A. Chadha, A. Sheth, and A. Das, "The troubling emergence of hallucination in large language models - an extensive definition, quantification, and prescriptive remediations," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 2541–2573.

[2] F. Liu, Y. Liu, L. Shi, Z. Yang, L. Zhang, X. Lian, Z. Li, and Y. Ma, "Beyond functional correctness: Exploring hallucinations in llm-generated code," 2024.

[3] J. Wang, X. Luo, L. Cao, H. He, H. Huang, J. Xie, A. Jatowt, and Y. Cai, "Is your ai-generated code really safe? evaluating large language models on secure code generation with codeseceval," *arXiv preprint arXiv:2407.02395*, 2024.

[4] L. Kumar, V. Singh, S. Patel, and P. Mishra, "Empowering sw security: Codebert and machine learning approaches to vulnerability detection," in *Proceedings of the 21st International Conference on Natural Language Processing (ICON)*, 2024, pp. 399–407.

[5] Y. Dong, J. Ding, X. Jiang, G. Li, Z. Li, and Z. Jin, "Codescore: Evaluating code generation by learning code execution," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 3, pp. 1–22, 2025.

[6] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *Advances in neural information processing systems*, vol. 28, 2015.

[7] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[8] J. Luo, T. Li, D. Wu, M. Jenkin, S. Liu, and G. Dudek, "Hallucination detection and hallucination mitigation: An investigation," *arXiv preprint arXiv:2401.08358*, 2024.

[9] Y. Tian, W. Yan, Q. Yang, X. Zhao, Q. Chen, W. Wang, Z. Luo, L. Ma, and D. Song, "Codehalu: Investigating code hallucinations in llms via execution-based verification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 24, 2025, pp. 25 300–25 308.

[10] P. Manakul, A. Liusie, and M. Gales, "Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models," in *Proceedings of the 2023 conference on empirical methods in natural language processing*, 2023, pp. 9004–9017.

[11] s. chen, Y. Zhao, J. Zhang, I.-C. Chern, S. Gao, P. Liu, and J. He, "Felm: Benchmarking factuality evaluation of large language models," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 44 502–44 523.

[12] V. Agarwal, Y. Pei, S. Alamir, and X. Liu, "Codemirage: Hallucinations in code generated by large language models," *arXiv preprint arXiv:2408.08333*, 2024.

[13] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, pp. 1536–1547.

[14] P. Zhou, W. Shi, J. Tian, Z. Qi, B. Li, H. Hao, and B. Xu, "Attention-based bidirectional long short-term memory networks for relation classification," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2016, pp. 207–212.

[15] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2999–3007.

[16] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: representing model uncertainty in deep learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16, 2016, p. 1050–1059.

[17] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[18] N. Jiang, Q. Li, L. Tan, and T. Zhang, "Collu-bench: A benchmark for predicting language model hallucinations in code," *arXiv preprint arXiv:2410.09997*, 2024.

[19] Z. Kasner, S. Mille, and O. Dušek, "Text-in-context: Token-level error detection for table-to-text generation," in *Proceedings of the 14th International Conference on Natural Language Generation*, 2021, pp. 259–265.

[20] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[21] Y. Chai, Y. Fang, Q. Peng, and X. Li, "Tokenization falling short: On subword robustness in large language models," in *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024, pp. 1582–1599.

[22] W.-C. Wang and J. Mueller, "Detecting label errors in token classification data," in *NeurIPS 2022 Workshop on Interactive Learning for Natural Language Processing (InterNLP)*, 2022.

[23] S. B. Hossain, N. Jiang, Q. Zhou, X. Li, W.-H. Chiang, Y. Lyu, H. Nguyen, and O. Tripp, "A deep dive into large language models for automated bug localization and repair," *Proc. ACM Softw. Eng.*, vol. 1, 2024.

[24] P. Puchert, P. Poonam, C. van Onzenoodt, and T. Ropinski, "Llmmaps–a visual metaphor for stratified evaluation of large language models," *arXiv preprint arXiv:2304.00457*, 2023.

[25] Y. Tao, Y. Qin, and Y. Liu, "Retrieval-augmented code generation: A survey with focus on repository-level approaches," *arXiv preprint arXiv:2510.04905*, 2025.